Final Project Report for CS 175, Winter 2019
Project Title: Hand Pose Estimation: How Many Fingers Am I Holding Up?
Project Number: 40

Student Name(s)
Dominic Langmesser, 53645159, dlangmes@uci.edu
Matthew Sanchez, 58578297, matthes3@uci.edu

1. Introduction and Problem Statement

The problem we addressed in this project is to detect the locations of the 21 hand key points as well as left/right orientation within a hand image. This type of software could be used to recognize sign language, for gesture controls of applications or AR applications in the future. To simplify the problem for our project we built an algorithm to determine the number of fingers that can be seen in the image. This is a simpler problem than the CMU research tracking all 21 key points in the hand.

2. Related Work

Hand pose estimation is a popular study in Computer Vision that has attracted lots of attention in the field, and thus we had a lot of papers to take inspiration from. Other approaches employed overlapping input spaces, bootstrapping, data augmentation, etc.

We knew we would want to use a Convolutional Neural Network because according to a study done by Markus Oberweger in 2015, even a standard architecture performs remarkably well. This study used convolutional neural networks with overlapping inputs, as shown in Fig. 1. They found that using "multiple input regions centered on the initial estimates of the joints, with very small pooling regions of the smaller input regions, and larger pooling regions for the larger input regions" increased localization accuracy and speed [2]. The smaller regions and the larger regions provided significant improvements in accuracy and contextual information, respectively. This provided us insight that we can use "non-conventional" architectures, and still get favorable results.

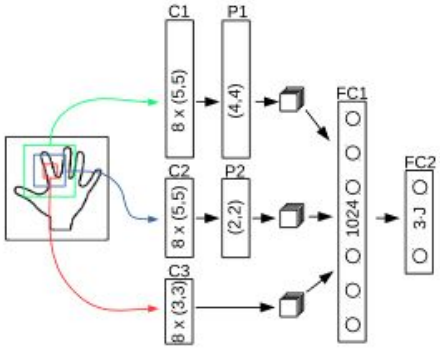

Fig. 1: Overlapping Input Architecture

Another research paper done in 2016 by Dmytro Mishkin studied the impact of recent CNN architectures and their results on object categorization problems. All of their tests were trained on the ImageNet dataset, which we discussed in class, with 1000 object categories. The size of their data is much larger than ours, with 1.2M training data, 50K validation data, and 100K test data, however we still gained good intuition on the different network configurations testing on in this study. Furthermore, this paper guided us on how to tune our hyperparameters and data to allow us to achieve deeper neural networks. First, they found that a 128x128 pixel image is "sufficient to make qualitative conclusions about optimal network structure" for Convolutional Neural Networks [1]. A graph made by Mishkin and his team is shown in Figure 2 to illustrate this idea. Second, we found that although larger filter sizes can lead to accuracy gain, the network gets saturated quickly. We evaluated the performances on varying filter sizes, and found that the network would benefit more from smaller filter sizes.

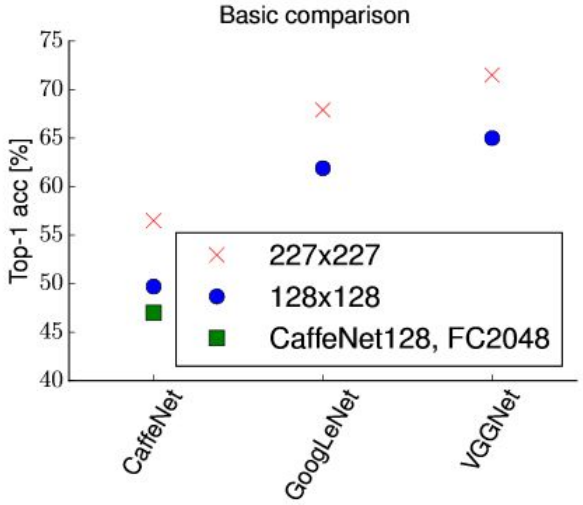

Figure 2: Impact of image and network size on top-1 accuracy (Mishkin)

3. Data Sets

We used the CMU panoptic handDB dataset, specifically the synthetic poses portion. We took images from synth2 and synth3 since they were similar to each other, and were focused on the hand rather than a person. At first, we were going to predict the labels that were given to us by the dataset, but we soon found out that this way out of the scope of our resources. The dataset is meant to be used to predict the 21 keypoints in the hand, if the keypoint is visible in the image, and if the image is of a left hand or not. The labels are arranged in a 2 item dictionary with the first key being "hand_pts" corresponding to a 21x3 list, and the second key being "is_left", whose value is 0 or 1. The 21x3 list contains the 21 keypoints' xy coordinate in the given 368x368 image, as well as a binary value to show if the keypoint is visible in the image. The coordinates have an order of magnitude of -11 and an int that could be anywhere from 0-368.

After processing the data, we realized just how optimistic we were with our ability with predicting all 21 keypoints. With our past experience of classifying images to 10 possible labels, we found it difficult and computationally expensive to continue this method of classification. For this reason, we decided to predict how many fingers are being held up or mostly visible in the image. Table 3 depicts our labeling method in which we counted how many fingers are prominent in the image. For example, image B is labeled as 1 due to the thumb being extended and the rest being curled. Image C is labeled as 2 because the ring and pinkie finger are the most visible in the image, even though the pointer finger is partially visible behind the hand.

Although we simplified the scope of our project, we acknowledge the disadvantages of using self-labeled data due to the inevitable bias and inconsistency issues. We attempted to use a concrete method of labeling the data, where we would count the number of proximal phalanges, the lower third of a finger, visible in the image, however this was hard to implement in practice. For instance, image B has all 5 visible, but we would only want to count the thumb since it is more prominent than the others. Quantifying this concept of "promimentness" is hard to estimate for each image as some can be partially hidden but still extended. The second issue with our data is size. The two folders of images we are using, provided us with 5,591 images. However, since we switched from given labels to self-labels relatively late in the project window, it was difficult to get a sizeable dataset labeled with our method.

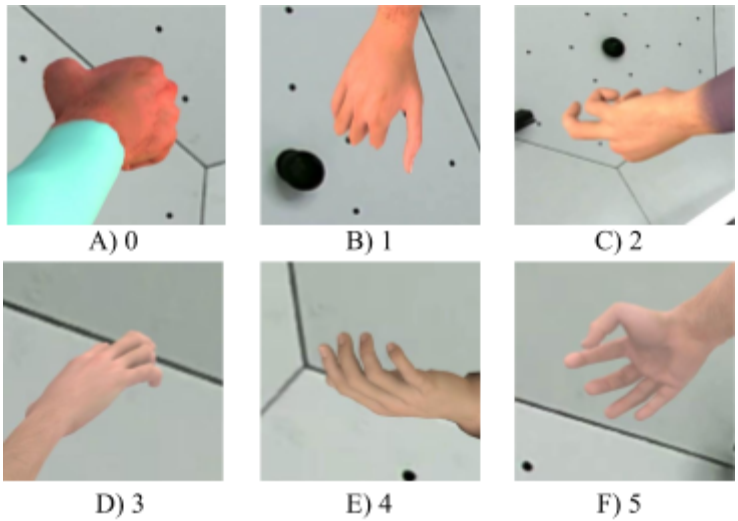

Table 3: How we labeled images in our data set

4. Description of Technical Approach

We spent a lot of time organizing the data and figuring out how to read the data in the correct way to get VGG19 to properly run. This took much more time than expected. However, training times have been much shorter than expected and we have made up much of the time that was lost building the algorithm to properly read in the data and labels. We attempted to address the lack of data situation by adding a horizontal flip to augment the training data. We looked into other types of augmentation such as random crop but decided not to use this due to the possibility of cropping out some of the fingers needed to count.

We utilized VGG19 for the first portion of the classification problem. We chose VGG as it was most similar to the work we have done in some of the assignments we did in class and thought we could maximize our results through fine tuning. We used pytorch's built-in VGG19 as the model husk and reimplemented the last section, the "classifier" section, to fit our data. We used fully connected layers and 50% dropout along with ReLU to first shrink the 25,088 outputs down to a manageable 4096 and eventually down to our 6 outputs. The results so far from VGG have been much worse than expected. The best results we have been able to achieve through adjusting the classification layers as well as hyperparameter tuning has been just under 50% accuracy on test and validation sets. Through research online as well as some of our own testing it appears small batch sizes greatly hinder this type of network. This, however, is an unavoidable problem with our current testing setup. We only have 4gb of VRAM on the card used for training and have been forced to limit the batch sizes below a size of 8. Through research we have found many people using the largest batch sizes they can afford to get the best results, most if not all successful results we found used batch sizes greater than 64, with some as high as 1024.

This led us to reconfiguring our code to run on Google Colab. This did not yield the results we had hoped either, as VGG19 still ran out of GPU memory at batch size of 64. However, we were able to yield better results after all thanks to the ability to train two models simultaneously and the increased maximum batch size up to 16 on colab. We used both from scratch models and pretrained models to see if this made a difference. The models from pytorch VGG are trained on ImageNet. The pretrained versions did slightly better on the shallow models (vgg11 & vgg13) and did much worse than from scratch models on the deeper nets (vgg16 & vgg19).

4. Software

a) Code/Scripts we have written

First, we wrote customDataset.py, with the template given to us in Professor Xie's CS175 GitHub file, pytorch-tutorial.ipynb. In it, we loaded the data, separating the .jpg and .json files with the python library glob. For each image, we resized it to 224x224 from its default size of 368x368 to align with our VGG architectures, and stored their RGB pixel values in an array. The

temporary array was built to zero-center and normalize the data for pre-processing. This was built in order to predict with all 22 classifiers, the 21 hand keypoints and left or right hand.

b) Code/Scripts written by others

The bulk of our train_model function in our VGG_colab notebooks are taken from a PyTorch tutorial online [3]. We also used much of what was given to us from assignment 3 during the quarter to assess accuracy.

5. Experiments and Evaluation

For VGG experimentation we set our notebook up in such a way we could easily swap out the VGG models (11-19) for comparisons. We used three separate data sets, 'train', 'val', and 'test' in different image folders to easily read in the data sets using torch's image folder data-loader. We had to manually label all of the data and separate the images into the corresponding folders, this led us to have very little data to work with as it was time consuming to hand label the data. We believe we could have achieved much better results had we thousands of data images to work with. We hand labeled a few hundred images and separated these into the three train/val/test categories with 250 train, 125 val and 125 test. We believe inconsistency in hand labeling could have led us to lower accuracy numbers as well. Some of the images are difficult to label as we used the synthetic generated hands. We used the synth dataset as the images were all the same size and all the hands were already centered in frame, this allowed us to center-crop the images and still be sure to get the hand data we needed.

Model	Loss	Pretrained	Val Acc (best)	Test Acc (best)	Epochs (best)
VGG-19_bn	SGD(lr=0.001)	no	0.46	0.44	125
VGG-19_bn	SGD(lr=0.001)	yes	0.33	0.31	50
VGG-16_bn	SGD(lr=0.001)	no	0.38	0.41	75
VGG-16	SGD(lr=0.01)	yes	0.38	0.34	25
VGG-16	SGD(lr=0.01)	no	0.44	0.42	25
VGG-13_bn	SGD(lr=0.01)	yes	0.47	0.505	30
VGG-13_bn	SGD(lr=0.001)	no	0.45	0.47	125
VGG-11_bn	Adam(lr=0.01)	no	0.39	0.33	10
VGG-19_bn	Adam(lr=0.1)	no	0.41	0.33	10

Table 2: Detailed VGG accuracies

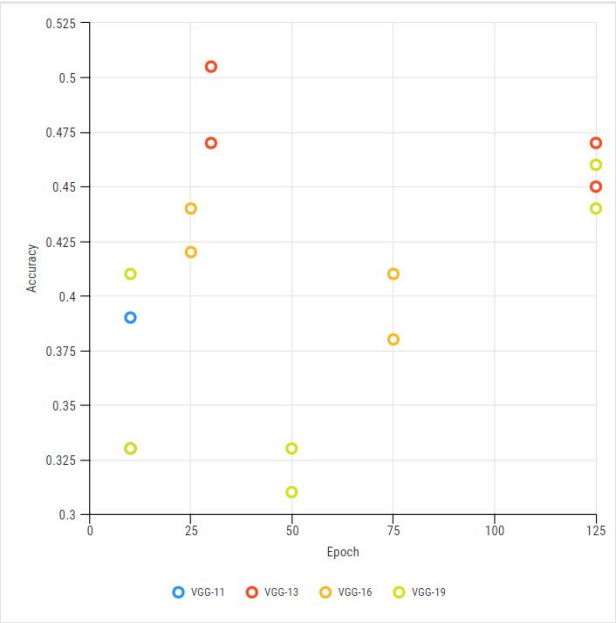

Figure 3: VGG accuracies

We also attempted to implement the ResNet architecture, however memory constraints, even on Google Collab, did not allow us to progress very far. Our original plan was to ensemble VGG and ResNet and to have them "vote" on which prediction had a higher confidence level. We included our ResNet notebook in the zip file for reference.

6. Discussion and Conclusion

We certainly both learned a lot from this project. We attempted to tackle a problem that was out of scope for our time limit in this quarter. We tried to adjust the scope down to something manageable with little time remaining. We learned how difficult and time consuming it can be to manually label training and test data without bias and with reasonable variety in data.

One of the main issues we've run into is naive versions of our model simply outputting the most common label for most if not all of the outputs. For our current training data 5 fingers is the most common output, so for many of our training sessions the model will simply output 120-125 '5's for the 125 test data points. We found that models that we over-fit actually avoided this problem. However, over-fit models did not perform well on the val and test data either as it was over-fitting to the training data. We believe a much larger dataset combined with larger batch-sizes and k-fold cross validation could solve these problems and greatly increase accuracy results.

7. References

[1] Mishkin, D., N. Sergievskiy, and J. Matas. "Systematic evaluation of CNN advances on the ImageNet. arXiv 1606: 02228 [cs]." *arXiv preprint arXiv:1606.02228* (2016).

[2] Oberweger, Markus, Paul Wohlhart, and Vincent Lepetit. "Hands deep in deep learning for hand pose estimation." *arXiv preprint arXiv:1502.06807* (2015).

[3] https://pytorch.org/tutorials/beginner/transfer_learning_tutorial.html